

COMPRESSING MULTI-COMPONENT DIGITAL MAPS USING JBIG2

Pasi Fränti, Eugene Ageenko, Pavel Kopylov and Sami Gröhn

Department of Computer Science, University of Joensuu
Box 111, FIN-80101 Joensuu, FINLAND

ABSTRACT

Digital maps can be stored and distributed electronically using compressed raster image formats. In this work, we study how the latest JBIG2 standard can be used for storing the map images. Image tiling must be performed to support direct access to partial images in the compressed file. This can be implemented by storing each image block as its own segment. Another problem is how to store the multiple image components. There are alternative ways to solve this problem by utilizing the features of JBIG2 in creative ways.

Keywords: Map images, real-time applications, personal navigation, image compression, spatial access, JBIG2.

1. INTRODUCTION

Digital maps can be stored and distributed electronically using compressed raster images. We consider images from the *Topographic Map Series 1:20 000* of Finland [5]. The images consist of several logical binary layers, which together form the computer-generated color image. The main problem for the use of the images is their huge size. For example, a single map sheet of 5000×5000 pixels representing a single map sheet of 10×10 km² requires about 12 megabytes.

Another highly desired feature is *spatial access* to the image. It enables the user to operate directly on the compressed data without retrieving the entire image. It makes possible an efficient retrieval and decompression of the desired parts of the image with high precision. Spatial access can be supported by tiling, i.e. dividing the image into fixed size rectangular blocks [1]. The blocks are then compressed independently by any compression method. The main problem is to combine the spatial access with efficient compression.

JBIG [4] and JBIG2 [3] are the latest standards for binary image compression. They use context-based statistical modeling with arithmetic coding, and result in 20:1 compression for typical map images. In addition to that, JBIG2 includes new useful features such as image segmentation, symbol dictionaries, and multiple pages. These new features are important when compressing multi-component images divided into smaller blocks for supporting spatial access. Especially the possibility of composing the image from multiple segments is highly useful.

In this work, we study how JBIG2 standard can be used for storing multi-layer map images. The image tiling can be implemented by storing each image block as its own segment.

A slightly more difficult problem is to store the multiple image components because the standard does not have any direct support for it. There are alternative ways to overcome this problem by utilizing the features of JBIG2 in creative ways.

We show by experimental results that the block division of 128×128 can be implemented with about 20% overhead if we follow the JBIG2 standard directly, and with about 6% overhead if we use the standard in a creative way. The biggest problem turns out to be that JBIG2 includes offsets of the data segments in the file header but the spatial locations of the segments are left in the data header, and not in the file header as desired.

2. DIGITAL MAP IMAGES FOR REAL-TIME NAVIGATION

Digital maps provide visual view on a given geographic location that can be used for applications dealing with spatial data, *geographic information systems* (GIS), or in *personal navigation*. We can select user-specific views of the map server for different applications. The main goal is to have the maps available in real-time without excessive computing resources, and independent of the location of the user.

Typical devices have limited memory resources and very narrow wireless communication channel (or no communication channels at all). The end-user cannot therefore store large map databases and complicated software for the database management. A much simpler but still useful approach is to provide the user with digital map images in compressed raster format [2].

We consider topographic maps provided by the *National Land Survey of Finland* (NLS) [5]. The maps are represented here as raster images of size 5000 × 5000, each map divided into several binary layers as shown in Fig. 1. The color of a pixel in the image is determined by checking the pixels in each layer. If the pixel is set as “foreground color”, the pixel is colored by the appropriate color assigned to that particular layer. Next, we define important properties that must be taken into account in the storage of the map images.

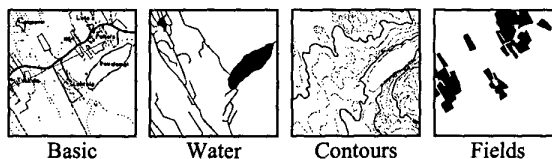


Figure 1: Sample multi-layer map image of size 500×500.

2.1 Compression

The images can be generated from the database beforehand and stored in the navigation device, or the maps can be requested on-line using any suitable communication channel. Because of the limited memory and channel resources, the images must be stored compactly and restored efficiently. Latest binary image compression standards JBIG and JBIG2 are good candidates for the compression task.

2.2 Multi-scale maps

The user must have the capability to browse the map with variable resolution. This can be achieved using (1) progressive compression method as presented in JBIG, or (2) by storing multiple maps representing the same area with various scale. The first approach has the same image re-scaled for several different resolutions. The compressed file contains the lowest resolution image followed by the higher resolution images in an increasing order.

The second approach has the advantage that the maps can be optimized for each resolution separately. For example, maps with different scale can have different colors and layout, see Fig. 2. The different scales are stored independently from each other, which adds 33% to the storage size compared to a single image of the highest resolution (assuming that the scale is decreased by a factor of 2).

The principle difference between these two approaches is that the maps with different scales have usually different level of details. In other words, some objects may be present in one scale but absent in another scale. On the other hand, if we would apply the resolution reduction for generating lower resolution image, the level of details remains the same as in the original image. Therefore, the maps with significantly reduced resolution would look messy.

2.3 Spatial access

Spatial access enables the user to operate directly on the compressed data. The actual viewing device can be very small, e.g. 200×200 pixels, and it is therefore important to have direct access to an image fragment in the compressed file without having to retrieve and decompress the entire image. If spatial access is supported, an image may be interactively browsed on the viewing device. When the image is scrolled, a new part of data is retrieved and decompressed on the fly. Thus, spatial access eliminates time delays caused by image decompression and transfer.

Spatial access is supported by dividing the image into fixed size rectangular blocks, and compressing the blocks independently. Pointers (*block indices*) are stored in the beginning of the compressed file to indicate the starting location of each block in the file, see Fig. 3. The blocks can then be decompressed in random order. The use of this kind of file organization is illustrated in Fig. 4.

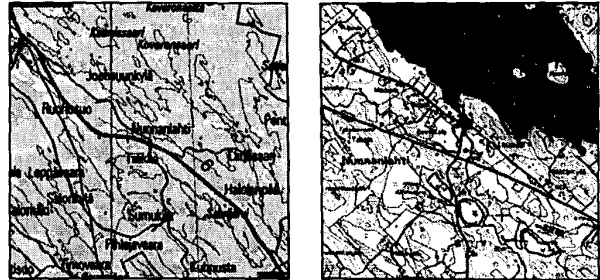


Figure 2: Sample images with the scales 1:80 000 and 1:20 000.

The block size is a trade-off between compression efficiency and decoding delay. If very small block size is used the desired part of the image can be reconstructed more accurately and faster. The compression, however, would be less efficient because of an increased learning cost and a less accurate probability model because of the block boundaries. The index table itself requires space and the overhead is relative to the number of blocks.

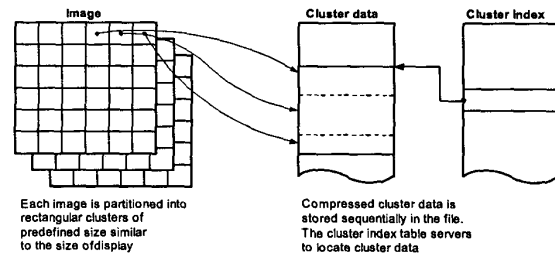


Figure 3: Diagram of the block decomposition.

3. IMPLEMENTATION OF SPATIAL ACCESS

The JBIG2 standard mainly defines the general file structure and the decoding procedure, but leaves some freedom in the design of the encoder. JBIG2 applies also pattern matching and text coding options but here we are not interested of them. Instead, the generic coding method of JBIG2 (similar to that of JBIG) is sufficient to us and we are more interested in the file organization itself.

The two most interesting features in JBIG2 standard are the multi-page option, and the image segmentation. The first one gives us an obvious but not necessarily the best way to store the multiple layers of the map images. The segmentation, on the other hand, gives us a tool to tile the image. We divide the image into $C \times C$ pixel blocks, and store each block as its own segment.

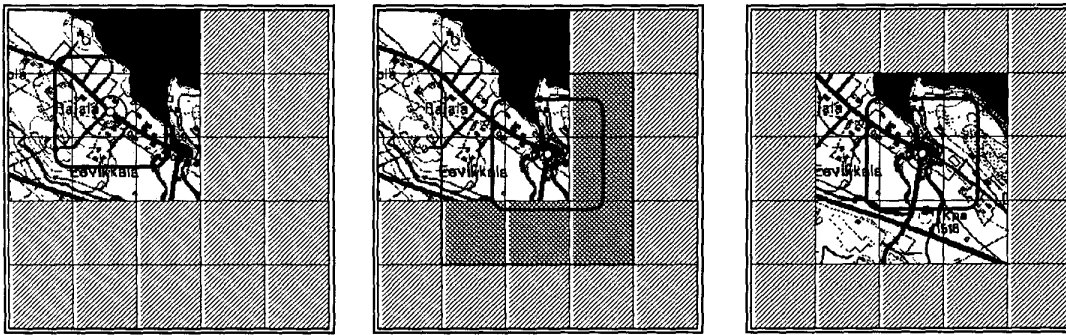


Figure 4: Use of method in clients device: Nine blocks are first decompressed and stored in the cache (left). Change of location is then registered (middle), and new image blocks are then decompressed and the view updated (right).

JBIG2 defines two alternative file organizations as shown in Fig. 5. The *segment* is a basic element of a JBIG2 file. It usually contains a header, which serves for identifying segment type, segment size as also page- and cross-segment dependencies. In *sequential organization*, segment data is located right after corresponding segment header. In *random-access organization* all the segment headers are in the beginning of the file. We use the random-access mode, and define the segments so that they cover the entire image.

In the following subsections, we study three different ways to organize the multi-layers images into a single file:

- *Variant 1:* All layers of a block in one segment
- *Variant 2:* Overlapping segments
- *Variant 3:* One layer per page

3.1 All layers in one segment

The *variant 1* stores all layers of a single image block subsequently into the same segment. The main advantage is that the overhead is minimized. The compressed data is organized so that the code from different layers follow each other subsequently. The layers are separated from each other by an extra escape sequence. Random access can be performed only to the first layer. This variant is not fully compatible with the JBIG2-standard, because a standard decoder would find only the first layer.

3.2 Overlapping segments

The *variant 2* stores every layer into its own segment. Thus, there will be L segments per every block and, therefore, several segments can represent the same image block. This alternative is compatible with the JBIG2 standard as any standard viewer is capable of viewing all layers although the image would most likely to be shown black-and-white. The disadvantage of this variant is that the overhead of the segment headers is L times as much as in the *variant 1*.

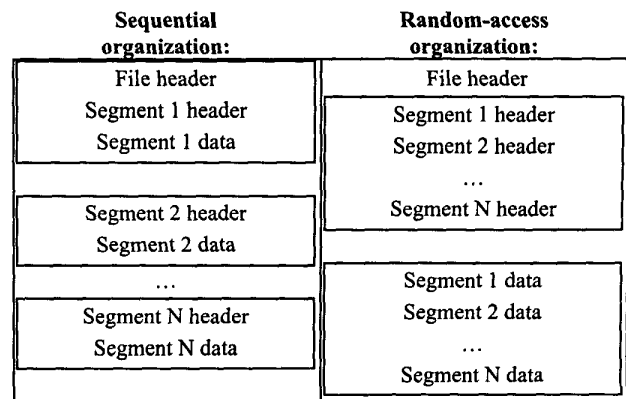


Figure 5: Two alternative file structures of JBIG2.

3.3 One layer per page

In *variant 3*, each layer is stored on a separate page. The pages will be combined into a single image by the viewer. This structure is compatible with the standard but the reconstruction of the color image from the layers is left to the user. The number of segments is the same as in the *variant 2*. In addition to that, the page overhead is multiplied by L , consisting of the page header and the "end of page sequence".

3.4 Summary

Image with L layers of size $X \cdot Y$ pixels, and the block size of $C \cdot C$ pixels results in the overhead shown in Table 1. The overhead is 13 bytes per file, 43 bytes per page, and 31 bytes per segment.

Table 1: Summary of overheads.

Segmentation:	Overhead (bytes):
No segmentation	$13 + 43 + 31 = 87$
Variant 1	$13 + 43 + 31 \cdot XY/C^2$
Variant 2	$13 + 43 + L \cdot 31 \cdot XY/C^2$
Variant 3	$13 + L \cdot 43 + L \cdot 31 \cdot XY/C^2$

The *variant 1* is the best choice if the file size is the only motive. There are, however, other factors that we must consider. For example, the user might want to have direct access to certain layers only. In this case, the *variant 1* would not be a feasible solution. Another factor might be the capability to interpret the images using standard JBIG2 decoder. In this case, the *variants 2* and *3* would be the only choices.

Much more serious problem is the way the direct access is implemented in JBIG2. It includes offsets of the data segments in the file header and, in this way, allows direct access to the compressed segments. Unfortunately the spatial location of the segments appear only in the header of the data segment. Usually this is not a problem because but in our case the situation is different because we need direct access to the compressed data on the basis of the location.

Even with the above restriction, it is still possible to identify the location of the segments by using a regular grid and store the blocks in a predefined order. The location of the segment can then be concluded from its position in the grid. The first drawback is that we must store all blocks including the empty ones. This causes unnecessary overhead when there are large empty areas in the image. The second drawback is that the fixed ordering excludes any dynamic insertion or deletion operations with the file structure, and thus, restricts possible use scenarios in dynamic map handling.

4. COMPRESSION RESULTS

We use five randomly chosen map images from the NLS *Basic Series* 1:20 000. For each image, we have four binary components divided into blocks of sizes 128×128, 256×256 and 5000×5000 (full image). Table 2 shows the compressed file sizes, and Table 3 the header overheads for each image separately (only non-empty blocks included). The overhead is 5.8% on average for *variant 1*, and about 15% for the other two variants in the case of block size 128×128.

The number of empty blocks (Table 4) is 35% (in case of 128×128) and 27% (in case of 256×256). This means that if we will use JBIG2, we must add these to the overhead, which would sum up to about 8% (128×128) and 20% (128×128) in case of the variants 2 and 3.

Table 2: Compressed file sizes from the compressed blocks for *variant 3*.

	<i>Full image</i>	256×256	128×128
Image 1	167,794	181,532	210,027
Image 2	888,046	980,088	1,069,347
Image 3	334,137	359,139	411,066
Image 4	738,560	775,048	866,001
Image 5	766,451	797,740	881,293
Total:	2,894,988	3,093,547	3,437,734

Table 3: Overheads of the different variants relative (%) to the compressed file.

	<i>Full image</i>	256×256	128×128
Variant 1	0.01 %	1.61 %	5.78 %
Variant 2	0.02 %	4.66 %	15.11 %
Variant 3	0.09 %	4.67 %	15.12 %

Table 4: Number of non-empty blocks in the image.

	<i>Full image</i>	256×256	128×128
Image 1	4	830	3763
Image 2	4	261	1666
Image 3	4	539	2590
Image 4	4	355	1607
Image 5	4	214	1438
From all:	0%	27%	35%

5. CONCLUSIONS

We have studied how JBIG2 standard can be used for storing multi-layer map images. The layer separation and block division can be implemented within the current standard with an overhead is about 20% (with block size 128×128) assuming that we follow the JBIG2 standard (variants 2 and 3), and store also the non-empty blocks. On the other hand, if we apply the standard in a creative way (variant 1), and omit the storage of the empty blocks (by importing the segment locations from the data segment to file header), the overhead would be about 6%.

To sum up, the standard is useful for storing multi-level map images but the inappropriate support for the direct access increases the overheads and limits the possibilities to create dynamic map handling. This means that we must either define own file format similar to JBIG2, or invent even more creative solutions in order to achieve direct access without excessive overhead.

6. REFERENCES

- [1] Ageenko E. and Fränti P., "Enhanced JBIG-based compression for satisfying objectives of engineering document management system", *Optical Engineering*, **37** (5), 1530-1538, May 1998.
- [2] Ageenko E. and Fränti P., "Compression of large binary images in digital spatial libraries", *Computers & Graphics*, **24** (1), 91-98, February 2000.
- [3] Howard, P.G., Kossentini, F., Martins, B., Forchammer, S, and Rucklidge, W. J., "The emerging JBIG2 standard". *IEEE Trans. Circuits and Systems for Video Technology*, **8** (7), 838-848, November 1998.
- [4] JBIG. ISO/IEC International Standard 11544 (1993) *ISO/IEC/JTC1/SC29/WG9*; also ITU-T Recommendation T.82. Progressive Bi-level Image Compression.
- [5] National Land Survey of Finland, Opastinsilta 12 C, P.O.Box 84, 00521 Helsinki, Finland. (http://www.nls.fi/index_e.html)